Deadlock and Fairness in Morphisms of Transition Systems

Hesselink, Wim H.

Published in:
Theoretical Computer Science

Publication date:
1988

Theoretical Computer Science 59 (1988) 235–257
North-Holland

DEADLOCK AND FAIRNESS IN MORPHISMS OF TRANSITION SYSTEMS

Wim H. HESSELINK

Department of Mathematics and Computer Science, Groningen University, 9700 AV Groningen, The Netherlands

Communicated by J. de Bakker
Received July 1986
Revised July 1987

Abstract. In order to investigate fair and deadlock-free implementations of processes, a formalization is proposed in which processes are elements of transition systems. Morphisms of transition systems are introduced as an approximate formalization of implementation of processes. A morphism $f: X \to Y$ shows deadlock if there is an implementing process $x \in X$ such that x has no transitions and that the implemented process $f(x) \in Y$ has transitions. The morphism is called strict if the implementing processes x provide all transitions of the implemented processes $f(x)$. Strict morphisms are used to clarify the problem of a universal domain of processes, in which bisimilar processes are identified.

To talk of fairness, the alphabet of actions is structured by a family $(B_j)_{j \in J}$ of subsets. Each subset B_j formalizes one component process which is to be treated fairly. Infinitely many components are admitted. A morphism $f: X \to Y$ is said to be fair if every execution sequence in the transition system X such that set B_j is enabled infinitely many times by the image sequence in Y activates set B_j infinitely many times. For a given transition system Y, we construct a family of fair and deadlock-free morphisms to Y which is universal in a well-defined sense. This construction is applied to the fair communicating merge of a family of processes.

0. Introduction

0.0. Unbounded processes

In this paper we propose a general but elementary formalism for semantic investigation of not necessarily terminating processes with possibly infinite sets of actions and communications. We do not develop a syntax for processes. In fact, for our purposes the mathematical language of sets and functions is adequate.

0.1. Transition systems

Processes are treated as nondeterministic systems. They are characterized as a state with a set of available transitions. Each transition is a pair that consists of an action and a resulting state. This suggests the domain equation $X = \mathcal{P}(A \times X)$, where X is the set of states (or processes), and A is the set of actions, and $\mathcal{P}(A \times X)$ is the powerset of the cartesian product of sets A and X. As it stands, this equation cannot be solved because of conflicting cardinalities. In [2], a topological solution is proposed and the powerset $\mathcal{P}(A \times X)$ is replaced by the set $\mathcal{P}_c(A \times X)$ of the

closed subsets of $A \times X$. In this way, however, certain interesting processes are forbidden. See the example below in Subsection 1.2. In particular, fairness aspects are endangered, cf. [15, p. 457]. In fact, a set of processes where a given action a happens eventually has a limit process where action a never happens. Thus, set closure may destroy fairness.

We adopt the following solution. Processes are described as elements of a transition system, which is a set X with a fixed transition function $s: X \to \mathcal{P}(A \times X)$, cf. [12]. Elsewhere, transition systems are called nondeterministic automata, or process graphs, or synchronization trees.

0.2. Morphisms of transition systems

Loosely speaking, a function f from a transition system X to a transition system Y is called a morphism of transition systems if every transition in X corresponds to some transition in Y. This has the effect that a process $x \in X$ may be considered as a safe but not necessarily active implementation of the image process $f(x)$ in Y.

Usually, a morphism f increases the degree of nondeterminacy, in the sense that not all transitions of $f(x)$ are made available by process x. Morphism f is called deadlock-free if x provides at least one transition whenever $f(x)$ has at least one transition. Morphism f is called strict if the implementing processes x provide all transitions of the implemented processes $f(x)$.

Strict morphisms are closely related to Park's concept of bisimilarity, cf. [16]. We use strict morphisms of transition systems to obtain a kind of tower of transition systems, which may be considered as a solution to the domain equation of Subsection 0.1.

0.3. Fairness and random predictive choice

The concept of fairness has been studied in many papers, cf. [0, 2, 4, 5, 7, 9, 13, 15, 16, 17], to mention only a small selection. Recently, Francez has written a monograph on the subject, cf. [6]. It has been known for a long time that fairness can be implemented by means of random predictive choices, cf. [0, 2, 5, 9, 17]. In [0, 13], it is shown that proof systems based on these random choices are complete with respect to correct termination of sequential programs with fair nondeterminacy. In the present paper we give a mathematical proof that, under our model assumptions, random choices are necessarily involved in any implementation of fairness.

0.4. Fair implementations

By fairness we mean strong fairness. A mechanism P is said to be fair with respect to some component Q of P if every execution sequence of P that enables Q infinitely often also activates Q infinitely often. Fair implementation seems to be a more important concept. An implementation P' of P is said to be fair with respect to component Q if every execution sequence of P' such that the corresponding execution sequence of P enables Q infinitely often itself activates Q infinitely often. This idea is formalized in the concept of fair morphisms of transition systems, cf. Section 4.

0.5. Components that are to be treated fairly

In most studies of fairness the competing components are modeled as arguments of a fair-merge operator or as alternative command sequences of an infinite loop. In that way it is not possible to specify that only certain components are to be treated fairly. In [9], in the context of sequential programming, we proposed a more flexible way to specify which program parts are to be treated fairly. The same idea is applied here, in the context of processes. In fact, we specify which subsets B_j of the alphabet of actions A are to be treated fairly. We admit infinitely many subsets. As suggested by one of the referees, this may have applications to programming languages with dynamic process creation.

0.6. A universal family of fair implementations, the fair communicating merge

Our main result (Theorem 5.4) can now be announced. Let Y be a transition system and let $T=(B_j)_{j\in J}$ be a family of subsets of alphabet A. We assume that the index set J is nonempty, and finite or countably infinite. For each ordinal number λ, we construct a transition system Z^{λ}_{pat} where "pat" stands for patient. Z^{λ}_{pat} consists of pairs (y, m) where $y\in Y$ and $m=(m_j)_{j\in J}$ is a vector of ordinal numbers less than λ. The vector m is supposed to be patient, a condition which implies that the components m_j are large enough so that deadlock can be avoided. In the case that the set J is finite, the condition of patience is fairly well known, cf. [5, 0].

Theorem 5.4. (a) *If $\lambda\geq\text{card}(J)$, the projection $p_\lambda: Z^{\lambda}_{\text{pat}}\to Y$ is a T-fair, surjective, deadlock-free morphism.*

(b) *If $f: X\to Y$ is a T-fair morphism, there is an ordinal number λ and a morphism $h: X\to Z^{\lambda}_{\text{pat}}$ such that $f=p_\lambda\circ h$.*

Assertion (b) is a universality property. Roughly speaking, it says that every fair implementation X of Y is an implementation of some implementing transition system Z^{λ}_{pat}. It also follows that if λ is infinite, then every fair execution sequence in Y is an image of an execution sequence in Z^{λ}_{pat}. Finally, in Section 6, we will show how the fair communicating merge can be treated with our methods.

1. Processes and morphisms

1.0.

This section contains the basic definitions. Some easy examples are provided.

1.1. Transition systems

Let A be an arbitrary set. The elements of A are called *actions.* We define a *transition system* over A to be a pair (X, s), where X is a set and s is a set-valued function $s: X\to\mathcal{P}(A\times X)$. Here $\mathcal{P}(A\times X)$ is the powerset of the cartesian product of the sets A and X. The elements of X are called *processes.* The fact that the set

$s(x)$ contains a pair (a, y) is expressed by saying that action a signals a *transition* from process x to process y. This fact is denoted by $a: x \to y$. The function s is called the *transition function*. If no ambiguity can arise, we speak of the transition system X instead of (X, s). Henceforward we only consider transition systems over a single fixed set of actions A.

Remark. A transition system X can be viewed as a possibly infinite directed graph with edges labeled by elements of A, cf. [3, Subsection 1.2.2]. We do not use this point of view in definitions or proofs.

1.2. Propagation

A process x in a transition system X is said to be *empty* if the set $s(x)$ is empty. We define an *execution path* of a process x to be a finite or an infinite sequence of pairs $(a_i, x_{i+1})_{i \in I}$ such that $(a_i, x_{i+1}) \in s(x_i)$ for every index $i \in I$. Here we always use $x_0 = x$. The set I is supposed to be an initial segment of $\mathbb{N} = \{i \mid i \geq 0\}$. So $I = \mathbb{N}$ or $I = \{i \mid 0 \leq i < m\}$ with $m \in \mathbb{N}$. An execution path is said to be *maximal* if the index set I is infinite, or if it is finite and the last process x_m is empty.

A process x in X is said to be *well-founded* if every execution path of x is finite. It is said to be *of finite depth* if there is a number $n \in \mathbb{N}$ such that every execution path of x has a length less than n.

Example. Let $A = \{a\}$ and $X = \{x_r \mid r \in \mathbb{N}\}$. Let the transition function s be given by

$$s(x_r) = \{(a, x_q) \mid q > 0 \wedge (q+1 = r \vee r = 0)\}.$$

The diagram in Fig. 1 may be useful. Process x_1 is empty. The processes x_r with $r > 0$ are of finite depth. Process x_0 is well-founded, but not of finite depth. This process cannot be described in the framework of [2]. For, in the setting of that paper, every well-founded process over a finite set A is of finite depth, cf. [10, Section 3.5].

1.3. Morphisms of transition systems

Let X and Y be transition systems with transition functions s_X and s_Y. A function $f: X \to Y$ is called a *morphism* of transition systems if, for every transition $a: x \to x'$ in the system X, there is a transition $a: f(x) \to f(x')$ in Y. In other words, it is required that for every process $x \in X$ we have an inclusion

$$\{(a, f(x')) \mid (a, x') \in s_X(x)\} \subset s_Y(f(x)).$$

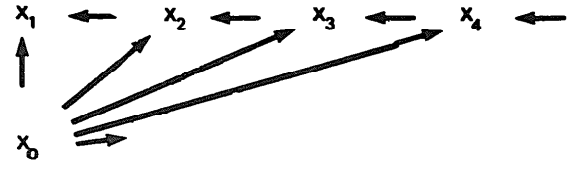

Fig. 1.

The morphism is said to be *strict* if for every process $x \in X$ we have

$$\{(a, f(x')) \mid (a, x') \in s_X(x)\} = s_Y(f(x)).$$

It is easily verified that the composition of morphisms $f: X \to Y$ and $g: Y \to Z$ is a morphism $g \circ f: X \to Z$. If both morphisms are strict, then so is the composition.

Example. Let $f: X \to Y$ be the function given by $f(x0) = y0$ and $f(x1) = y1$, where X and Y are the transition systems given in the diagram in Fig. 2. It is easy to verify that f is a morphism of transition systems. Process $x0$ may be considered as a safe implementation of $y0$, but the implementer has chosen not to provide any action b. At $x1$ the implementation shows deadlock, as will be formalized in Definition 1.5 below.

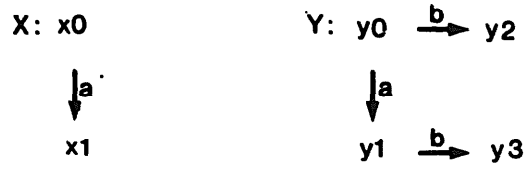

Fig. 2.

Remarks. (a) Morphisms and strict morphisms are also used in the more general context of nondeterministic data types, cf. [8].

(b) We do not define a formal concept of implementation of processes. However, we consider the existence of a morphism $f: X \to Y$ with $f(x) = y$ as a sufficient condition that process x is a safe implementation of process y. In fact, every action of x or of one of the descendants of x can be done by y or a corresponding descendant of y. It is a sufficient condition, not a necessary one, but a condition that is easily checked and that works in many cases. Compare [8, Section 5].

1.4. Subsystems

Let X be a transition system. A *subsystem* of X is defined to be a transition system U such that $U \subset X$ and that $s_U(x) \subset s_X(x)$ holds for every process $x \in U$. In other words, the injection map $U \to X$ must be a morphism of transition systems. A subsystem U is said to be a *full* subsystem if

$$\forall x \in U: \quad s_U(x) = s_X(x) \cap (A \times U).$$

Every subset U of X has a unique structure as a full subsystem.

Example. Let X be the transition system of the example in Subsection 1.2. Let U be the transition system with set $U = \{x_r \mid r \in \mathbb{N} \wedge r > 1\}$ and transitions $s_U(x_r) = \{(a, x_q) \mid 3 < q = r - 1\}$. System U is a subsystem of X but not a full subsystem. The processes x_2, x_3, x_4 are empty in U and nonempty in X.

1.5. Deadlock

We consider deadlock as a situation where a formal specification suggests certain actions to occur, but where the implementing process is unable to do anything. Therefore, a morphism of transition systems $f: X \to Y$ is considered to show deadlock if there is an empty process $x \in X$ such that $f(x)$ is nonempty in system Y. We define a morphism $f: X \to Y$ to be *deadlock-free* if for every empty process $x \in X$ the image process $f(x)$ in Y is also empty.

Example. Again, let X be the transition system of Example 1.2. Let Y be the transition system with only one process y and only one transition $a: y \to y$. The function $f: X \to Y$ with $f(x_r) = y$ for all indices r is a morphism with deadlock.

1.6. The living subsystem

Let $f: X \to Y$ be a morphism of transition systems. A subsystem U of X is said to be *living* with respect to f if the restriction $f|U$ is a deadlock-free morphism $f: U \to Y$.

A process $x \in X$ is said to be *living* with respect to f if it has some execution path $(a_i, x_{i+1})_{i \in I}$ in system X such that the image path $(a_i, f(x_{i+1}))_{i \in I}$ is a maximal path in system Y. Let X_{liv} be the full subsystem of X such that the set X_{liv} consists of the living processes x of X.

It is easy to verify that X_{liv} is a living subsystem of X. In fact, a process is empty if and only if the execution path of length 0 is maximal. Conversely, if U is a living subsystem of X, then U is a subsystem of X_{liv}. In fact, let $x \in U$ be given. In the transition system U we construct a maximal execution path $(a_i, x_{i+1})_{i \in I}$ of x. Since the morphism $f|U$ is deadlock-free, the image path is maximal in Y, so that x is in X_{liv}. This proves that X_{liv} is the biggest living subsystem of X with respect to f.

2. Observable transition systems

2.0.

In this section we develop a formalism that enables us to identify bisimilar processes and to put any family of processes together in one transition system.

2.1. Bisimilarity

Let X and Y be transition systems, say with transition functions s_X and s_Y. Processes $x \in X$ and $y \in Y$ are said to be *bisimilar* (notation $x \approx y$) if there is a binary relation $r \subset X \times Y$ with $(x, y) \in r$ and such that for every pair $(u, v) \in r$ and every action $a \in A$, we have

$$\forall u' \in X: \quad (a, u') \in s_X(u) \Rightarrow (\exists v' \in Y: (u', v') \in r \wedge (a, v') \in s_Y(v)), \quad \text{and}$$

$$\forall v' \in Y: \quad (a, v') \in s_Y(v) \Rightarrow (\exists u' \in X: (u', v') \in r \wedge (a, u') \in s_X(u)).$$

The concept of bisimilarity is due to Park, cf. [16]. It is also adopted in [14, 3]. It is easy to see that bisimilarity is reflexive, symmetric, and transitive.

2.2. Bisimilarity in one transition system

A *congruence* on a transition system X is defined to be an equivalence relation r on X such that, for every pair $(x, y) \in r$ and every transition $a: x \to x'$ in X, there is some transition $a: y \to y'$ in X with $(x', y') \in r$.

Clearly, every congruence r on X consists of pairs (x, y) that are bisimilar. Conversely bisimilarity defines an equivalence relation on X, which is a congruence. This shows that bisimilarity is the largest congruence.

Remark. In the more general context of nondeterministic data types, observable equivalence of values has been defined as the largest congruence, cf. [11, p. 29] and [8]. Therefore, bisimilarity is a special case of observable equivalence in that sense.

2.3. Quotient systems and observability

Let r be a congruence on a transition system X. The *quotient* system X/r is defined as the set X/r of the equivalence classes

$$x/r = \{y \in X \mid (x, y) \in r\}$$

together with the transition function s' given by

$$s'(x/r) = \{(a, y/r) \mid (a, y) \in s(x)\}.$$

The function s' is well-defined. This follows from the defining condition for congruences, cf. Definition 2.2.

The transition system X is said to be *observable* if bisimilarity is the identity relation on X. In the general case, if $\approx$ is the bisimilarity relation on X, then the quotient system $X/\approx$ is observable, cf. [8]. Therefore, $X/\approx$ is called the *observable quotient* of X.

Remark. Usually we are more interested in the observable quotient than in the original system. Most constructions, however, yield systems that need not be observable.

2.4. Strict morphisms and bisimilarity

If $f: X \to Y$ is a strict morphism of transition systems, the graph $\{(x, f(x)) \mid x \in X\}$ of the function f is a bisimilarity relation, which implies that $x \approx f(x)$ holds for every process $x \in X$.

On the other hand, if r is a congruence relation on X, then the quotient function $q: X \to X/r$ given by $q(x) = x/r$ is easily seen to be a strict morphism. In particular, the quotient function from X to its observable quotient $X/\approx$ is a strict morphism.

Remark (*suggested by a referee*). Bisimilar processes need not be connected by a strict morphism. For example, the processes $x0$ and $y0$ in Fig. 3 are easily seen to be bisimilar, but there is no strict morphism between the two systems.

Fig. 3.

2.5.

Theorem 2.1. *Let X and Y be transition systems.*

(a) *If Y is observable, there is at most one strict morphism from X to Y.*

(b) *If X is observable, every strict morphism $f: X \to Y$ is an injective function.*

Proof. (a): Assume that f and g are two strict morphisms from X to Y. If $x \in X$, then both processes $f(x)$ and $g(x)$ in Y are bisimilar to process x in X, by Subsection 2.4. Since Y is observable, it follows that $f(x) = g(x)$. This proves that $f = g$.

(b): If $f(x) = f(x')$, then, by Subsection 2.4, both processes x and x' in X are bisimilar to process $f(x)$. Since X is observable, it follows that $x = x'$. This proves that function f is injective. □

2.6.

Theorem 2.2. *Let $\{X_i \mid i \in I\}$ be a set of transition systems. Then there is an observable transition system X with strict morphisms $f_i: X_i \to X$.*

Proof. One first forms the disjoint union $Z = \bigsqcup_{i \in I} X_i$ with the transitions inherited from the participating transition systems X_i. One verifies that the canonical injections $j_i: X_i \to Z$ are strict morphisms. One forms the observable quotient $X = Z/\approx$ with the quotient morphism $q: Z \to X$. The compositions $f_i = q \circ j_i$ are strict morphisms. □

2.7. The class of observable transition systems

Let us define an observable transition system X to be *contained* in an observable transition system Y if there exists a strict morphism $f: X \to Y$. By Theorem 2.1, this morphism is necessarily unique, and injective. Therefore, if we identify isomorphic transition systems, containment defines an ordering in the class of observable transition systems. By Theorem 2.2, the class of observable transition systems is directed in a very strong sense. It may be considered as a kind of tower.

If the alphabet of actions A has at least one element, there is no observable transition system that contains all others. In fact, let X be observable. As the cardinality of X is less than the cardinality of the powerset $\mathcal{P}(A \times X)$, there is a subset U of $A \times X$ with $s(x) \neq U$ for all processes $x \in X$. We form the transition system $Y = X \cup \{y\}$ where y is a symbol not in X, with the transition function s

given by $s|X = s_X$ and $s(y) = U$. Since X is observable and $s(y) \neq s(x)$ for every process x in X, the system Y is observable. It is clear that Y is not contained in X. This result implies that the observable transition systems form a proper class in the sense of set theory.

2.8. Operators on processes

In the observable transition systems we can define addition of processes and prefixing with actions, cf. [14, 3]. For simplicity we only consider these two operations. Merge operations will be discussed in Section 6.

If x and x' are processes in an observable transition system X, the *sum process* $y = x + x'$ is specified by $s(y) = s(x) \cup s(x')$. If such a process y exists in X, then it is unique. It always exists in some observable transition system that contains X. If a is an action in A, the *prefixed process* $z = ax$ is specified by $s(z) = \{(a, x)\}$. Again, if z exists, then it is unique. It always exists in some containing system.

3. Fairness and ordinal functions

3.0.

In this section we investigate fairness under laboratory conditions. Recall that fairness means strong fairness throughout the paper. The main result is the relation with ordinal functions. We use the same methods as [13], but we characterize arbitrary fair paths, and not only fairly terminating ones.

We start with concept analysis. Let P be a mechanism that contains a number of concurrent components Q_j with $j \in J$. Mechanism P is said to be fair with respect to component Q_j if every execution sequence of P that enables Q_j infinitely many times also activates Q_j infinitely many times. In our process formalism the mechanism and its components disappear, and are replaced by one process x. In order to remember the component from which an action originates, the actions are labeled. In this way the alphabet A is made bigger, and it is equipped with a family of subsets B_j such that set B_j consists of the actions of component Q_j. If $j \neq k$, an action in the intersection $B_j \cap B_k$ might be a synchronous communication between components Q_j and Q_k. Therefore, such intersections need not be empty. Process x is said to be fair if for every index j every execution path of x that enables set B_j infinitely many times also activates B_j infinitely many times. In Subsection 3.1 we will give slightly more general definitions. The greater generality is needed for the fair morphisms of Sections 4 and 5.

3.1. Fairness

Let X be a transition system over alphabet A. If $B \subset A$ and $V \subset X$, then an execution path $(a_i, x_{i+1})_{i \in I}$ is said to be *B-V-fair* if

$$\text{card}\{i \mid a_i \in B\} = \infty \vee \text{card}\{i \mid x_i \in V\} < \infty.$$

We define the *B-enabling subset* $X(B)$ of the transition system X by

$$X(B) = \{x \in X \mid \exists (a, x') \in s(x): a \in B\}.$$

An execution path is said to be *B-fair* if it is B-$X(B)$-fair. Usually we need the following generalization. Let $T = (B_j)_{j \in J}$ be a family of subsets of A. An execution path is said to be *T-fair* if it is B_j-fair for every index $j \in J$.

A process $x \in X$ is said to be *B-V-fair*, or *B-fair*, or *T-fair* if every execution path of x in X is B-V-fair, or B-fair, or T-fair, respectively. Finally, we define transition system X to be *B-V-fair*, or *B-fair*, or *T-fair* if every execution path in X is B-V-fair, or B-fair, or T-fair, respectively.

3.2.

Proposition 3.1. *Let $x \in X$ and $y \in Y$ be bisimilar processes. Let $T = (B_j)_{j \in J}$ be a family of subsets of A. Then process x is T-fair if and only if process y is T-fair.*

Proof. Assume that y is T-fair. Let $\xi = (a_i, x_{i+1})_{i \in I}$ be an execution path of process x. By means of induction one constructs an execution path $\eta = (a_i, y_{i+1})_{i \in I}$ of process y in Y such that $x_i \approx y_i$ for all indices $i \in I$. This bisimilarity implies that

$$\forall j \in J: \quad \{i \mid x_i \in X(B_j)\} = \{i \mid y_i \in Y(B_j)\}.$$

Since process y is T-fair, path η is T-fair. Therefore, path ξ is T-fair. This proves that x is T-fair. The other implication follows by symmetry. □

3.3. Ordinal functions

Let Ord denote the class of the ordinal numbers, cf. [19]. The class Ord is not a set, but every element $\lambda \in \mathrm{Ord}$ is a set, which satisfies $\lambda = \{\alpha \in \mathrm{Ord} \mid \alpha < \lambda\}$. We use $\omega \in \mathrm{Ord}$ to denote the first infinite ordinal, so it is equal to the set $\mathbb{N}$.

If X is a set, a function $g: X \to \mathrm{Ord}$ is called an *ordinal function* on X. Since the image $\{g(x) \mid x \in X\}$ is a set, it has some least upper bound λ, and the function g can be considered as a function $g: X \to \lambda + 1$ between ordinary sets. Here we use

$$\lambda + 1 = \lambda \cup \{\lambda\} = \{\alpha \mid \alpha \leq \lambda\}.$$

3.4.

Theorem 3.2. *Transition system X is B-V-fair if and only if there is an ordinal function g on X such that for every transition $a: x \to y$ in X with $a \notin B$ it holds that $g(x) \geq g(y)$, and that $g(x) > g(y)$ whenever $x \in V$.*

Proof. The sufficiency is well-known, cf. [0]. In fact, let g be an ordinal function as specified. Let $\xi = (a_i, x_{i+1})_{i \in I}$ be an execution path in X. Assume that

$$\mathrm{card}\{i \mid a_i \in B\} < \infty.$$

We can choose a number j such that $a_i \notin B$ holds for all indices $i > j$. For all indices $i > j$ we have $g(x_{i+1}) \leq g(x_i)$. The well-foundedness of the ordinal numbers implies

that there is a number $k>j$ with $g(x_{i+1})=g(x_i)$ for all indices $i>k$. Therefore, we have $x_i \notin V$ for all indices $i>k$. This implies that the execution path ξ is B-V-fair. This proves that the transition system X is B-V-fair.

Conversely, assume that X is B-V-fair. We construct three binary relations on the set X. Let the relation R consist of the pairs (y, x) such that there is a transition $a: x \to y$ with $a \in A \setminus B$. Let the relation R_0 be the transitive closure of R. Let the relation R_1 be given by

$$R_1 = \{(y, x) \mid \exists z \in V: (y, z) \in R_0 \wedge ((z, x) \in R_0 \vee z = x)\}.$$

An ordinal function g on X satisfies the conditions of the theorem if and only if the following two formulas hold:

(0) $\forall (y, x) \in R_0$: $g(y) \leqq g(x)$,

(1) $\forall (y, x) \in R_1$: $g(y) < g(x)$.

The relations R_0 and R_1 are transitive. It even holds that

$$\forall x, y, z \in X: \quad (z, y) \in R_1 \wedge (y, x) \in R_0 \Rightarrow (z, x) \in R_1.$$

Since transition system X is B-V-fair, relation R_1 is well-founded. In fact, suppose that $(x_j)_{j \in \mathbb{N}}$ is an infinite sequence with $(x_{j+1}, x_j) \in R_1$ for all indices j. By the definition of R_1, we get an infinite execution path $(a_i, y_{i+1})_{i \in \mathbb{N}}$ such that $a_i \notin B$ holds for all indices i and that the set of indices i with $y_i \in V$ is infinite. This contradicts fairness. It follows that every nonempty subset U of X has a nonempty socle

$$\mathrm{Soc}(U) = \{x \in U \mid \forall y \in U: (y, x) \notin R_1\}.$$

By transfinite induction we construct for all ordinal numbers α certain level subsets X_α of X, by putting

$$X_\alpha = \mathrm{Soc}\left(X \setminus \bigcup_{\beta<\alpha} X_\beta\right).$$

The sets X_α are all disjoint. For reasons of cardinality there is an ordinal number α such that X_α is empty. Let λ be the smallest ordinal with X_λ empty. Then X is the disjoint union of the nonempty sets X_α with $\alpha < \lambda$. There is a unique ordinal function g on X with

$$g(x) = \alpha \Leftrightarrow x \in X_\alpha.$$

We verify that g satisfies the above conditions (0) and (1). If $(y, x) \in R_1$ and $g(x) = \alpha$, then $x \in X_\alpha$ so that the definition of the socle implies $y \in \bigcup_{\beta<\alpha} X_\beta$ and hence, $g(y) < \alpha$. This proves condition (1).

Assume that $(y, x) \in R_0$. Put $\alpha = g(x)$. If $z \in X$ satisfies $(z, y) \in R_1$, then we have $(z, x) \in R_1$ so that $g(z) < \alpha$ by condition (1), and hence, $z \in \bigcup_{\beta<\alpha} X_\beta$. Therefore, if $y \in X \setminus \bigcup_{\beta<\alpha} X_\beta$, then we have $y \in \mathrm{Soc}(X \setminus \bigcup_{\beta<\alpha} X_\beta)$. This implies that $g(y) \leqq \alpha$, thus proving condition (0). □

3.5. Remarks

Let an ordinal function g as specified in Theorem 3.2 be called a *patience function.* It is easy to verify that the infimum of a set of patience functions is again a patience function. So there is a unique smallest patience function. In fact, the patience function constructed in the proof of Theorem 3.2 is the smallest one.

Theorem 3.2 is closely related to [13, Theorem 1], but it does not seem to be a direct consequence.

3.6. Remark

In Theorem 3.2, large ordinals may be needed. In fact, if $\emptyset \neq B \neq A$, then one can construct a finitely branching B-fair transition system X such that the smallest patience function g on X requires all countable ordinals.

Example. Choose actions $b \in B$, and $c, d \in A \setminus B$. By transfinite induction we form processes y_λ as follows. We use the notation of Definition 2.8. Let x be the empty process. Put $y_0 = bx$. If $\lambda = \mu + 1$, we let $y_\lambda = bx + cy_\mu$. If λ is a countable limit ordinal, we choose an enumeration $e: \omega \to \lambda$ and we form the processes $z_{\lambda,n} = dz_{\lambda,n+1} + cy_{e(n)}$, where $n \in \omega$ and $y_\lambda = z_{\lambda,0}$. This construction stops at the first uncountable ordinal. Let X be the transition system that contains the processes constructed in this way. The system X is finitely branching. The set $X(B)$ consists of the processes y_λ with $\lambda = 0$ or $\lambda = \mu + 1$. Therefore, every execution sequence of X contains at most finitely many processes of $X(B)$. This proves that X is B-fair. The smallest patience function g on X satisfies $g(y_\lambda) = \lambda$ for all countable ordinals λ. The transition system X can be considered as a safe and B-fair implementaion of the program

do $u = 1 \to b$: $u := 0$ {at x}

$[\!]$ $u \neq 0 \to c$: $u := 1$ {at some y_λ}

$[\!]$ $u \neq 0 \to d$: $u := 2$ {at some $z_{\lambda,n+1}$}

od.

The result can be compared with [1] where an analogous statement is proved for recursive ordinals only.

3.7. Remark

Transfinite ordinals are not needed in the following analogue of Theorem 3.2. The proof of this result is easy and may be left to the reader.

Fact 3.3. *An execution sequence $(a_i, x_{i+1})_{i \in \omega}$ of a process x_0 in a transition system X is B-V-fair if and only if there is a function $g: \omega \to \omega$ such that for every index $i \in \omega$ with $a_i \notin B$ it holds that $g(i) \geq g(i+1)$, and that $g(i) > g(i+1)$ whenever $x_i \in V$.*

4. Fair morphisms

4.0. Fair implementations

We start with concept analysis, as in Subsection 3.0. Assume that mechanism P with its components Q_j is to be implemented by mechanism P'. The implementation P' is said to be fair with respect to component Q_j if every execution sequence of P' such that the corresponding execution sequence of P enables Q_j infinitely many times itself activates Q_j infinitely many times. This idea is captured in the concept of fairness of morphisms.

4.1. Fairness of morphisms

Let $T=(B_j)_{j\in J}$ be a family of subsets of alphabet A, cf. Definition 3.1. A morphism of transition systems $f: X \to Y$ is said to be *T-fair* if for every execution path $(a_i, x_{i+1})_{i\in I}$ in X the image path $(a_i, f(x_{i+1}))_{i\in I}$ in Y is T-fair.

Remarks. (a) One verifies that every path in X with a T-fair image path in Y is itself T-fair. In fact, for every index $j\in J$ we have the inclusion

$$\{i\in I \mid x_i \in X(B_j)\} \subset \{i\in I \mid f(x_i)\in Y(B_j)\}.$$

It follows that if morphism $f: X \to Y$ is T-fair, then X is T-fair. The converse implication is not true, cf. Example 4.2 below.

(b) On the other hand, it is clear that if Y is T-fair, then morphism $f: X \to Y$ is T-fair.

(c) It is easy to see that morphism $f: X \to Y$ is T-fair if and only if X is B_j-V_j-fair for every index $j\in J$, where $V_j = f^{-1}(Y(B_j))$.

4.2. Example

Let the alphabet of actions be $A=\{a, b\}$. Let Y be the transition system with set $Y=\{y\}$ and transition function s_Y given by $s_Y(y)=\{(a, y), (b, y)\}$ (see Fig. 4). System Y is observable with one process $y = ay + by$. We consider fairness with respect to the family $T=(\{a\},\{b\})$. System Y is not T-fair since action b is always enabled and there is an infinite execution path that never executes b. Let X be the subsystem of Y with set $X = Y$ and transition function s_X given by $s_X(y)=\{(a, y)\}$. This subsystem is T-fair as action b is never enabled in X and action a is always taken in an infinite execution path in X. The injection morphism from X into y is not T-fair since action b is always enabled in Y and never taken by X.

Fig. 4.

A T-fair implementation of Y can be obtained as follows. Let transition system Z be the set of the integers with the transition function s_Z given by

$$s_Z(m) = \{(a, n) \mid 0 \leq m \wedge n < m\} \cup \{(b, n) \mid m \leq 0 \wedge m < n\}.$$

System Z is a kind of pendulum. The degree of nondeterminacy is very high. In Fig. 5 we use diagonal arrows to indicate transitions which leap over one or more nodes. One verifies that the unique function $Z \to Y$ is a T-fair morphism. This example is a slight variation of the following general construction, which goes back to [5].

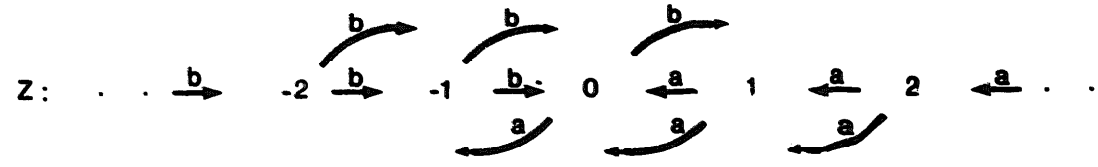

Fig. 5.

4.3. The fair implementation system Z^λ

Let a transition system Y and a family $T = (B_j)_{j \in J}$ be fixed. We form a family of T-fair morphisms $p_\lambda : Z^\lambda \to Y$ such that every T-fair morphism $f : X \to Y$ has a factorization over some system Z^λ with λ sufficiently large. For the moment, the danger of deadlock is deliberately ignored.

Let λ be a given ordinal number. The set Z^λ consists of the pairs (y, m) where y is a process in Y, and $m = (m_j)_{j \in J}$ is a vector of ordinal numbers less than λ, indexed by elements of J. Thus Z^λ is the cartesian product $Z^\lambda = Y \times \lambda^J$. The set Z^λ is made into a transition system by specifying that an element (y, m) has the transition set

$$s(y, m) = \{(a, (y', m')) \mid (a, y') \in s(y) \wedge \forall j \in J:$$
$$a \notin B_j \Rightarrow (m_j \geq m'_j \wedge (y \in Y(B_j) \Rightarrow m_j > m'_j))\}.$$

We let $p_\lambda : Z^\lambda \to Y$ be the projection function with $p_\lambda(y, m) = y$. It is clear that p_λ is a morphism of transition systems.

4.4.

Theorem 4.1. (a) *The morphisms $p_\lambda : Z^\lambda \to Y$ are T-fair.*

(b) *Let $f : X \to Y$ be a T-fair morphism. Then there is an ordinal number λ with a morphism of transition systems $h : X \to Z^\lambda$ such that $f = p_\lambda \circ h$.*

Proof. (a): Fix λ. Let $k \in J$ be given. By Remark 4.1(c), it suffices to prove that Z^λ is B-V-fair, where $B = B_k$ and $V = p_\lambda^{-1}(Y(B_k))$. Let the ordinal function g on Z^λ be defined by $g(y, m) = m_k$. We verify that g satisfies the conditions of Theorem 3.2. Let $a : z \to z'$ be a transition in Z^λ with $a \notin B$. Assume that $z = (y, m)$ and that

$z' = (y', m')$. Then we have $a: y \to y'$ in Y, and $g(z) = m_k \geq m'_k = g(z')$. Moreover, if $z \in V$, then $y \in Y(B_k)$ and hence, $g(z) = m_k > m'_k = g(z')$. By Theorem 3.2, this proves that Z^λ is B-V-fair.

(b): For every index $j \in J$, the transition system X is B_j-V_j-fair, where $V_j = f^{-1}(Y(B_j))$. By Theorem 3.2, there exist ordinal functions g_j on X such that, for every transition $a: x \to x'$ in X with $a \notin B_j$, it holds that $g_j(x) \geq g_j(x')$ and that $g_j(x) > g_j(x')$ whenever $f(x) \in Y(B_j)$. For cardinality reasons we can choose an ordinal number λ such that $g_j(x) < \lambda$ holds for every process $x \in X$ and every index $j \in J$. We define the function $h: X \to Z^\lambda$ by

$$h(x) = (f(x), (g_j(x))_{j \in J}).$$

The conditions on the functions g_j imply that $h: X \to Z^\lambda$ is a morphism of transition systems. It is clear that $p_\lambda \circ h = f$. □

4.5. Remark

The morphisms of Theorem 4.1 represent possibly deadlocking implementations, as also considered in [1]. In our view, Theorem 4.1(b) shows that random predictive choices are inevitable in any operational description of fairness. In [4] it is claimed that (weak) fairness can be obtained without random predictive choices. We have the impression, however, that the method of [4] to obtain all fair execution sequences is based on making unboundedly many steps at the same time. This is more or less the same as predictive choice. We come back to this in the last paragraph of Subsection 6.3 below.

5. Fairness and liveness

5.0. Introduction

In the transition systems Z^λ of Subsection 4.3 the ordinal vectors m are used as priority vectors. If $z = (y, m)$, then the set B_j can be enabled not more than m_j times before it must be activated. Deadlock occurs if $m_j = 0 = m_k$ and both sets B_j and B_k are enabled but not with a common action. This shows that the morphism $p_\lambda: Z^\lambda \to X$ need not be deadlock free.

The biggest living subsystem Z^λ_{liv}, cf. Definition 1.6, depends on the precise structure of system Y and family T. Fortunately, however, system Z^λ has a sufficiently large living subsystem Z^λ_{pat} which can be described explicitly. In the case of finitely many processes, the main ideas in the construction of Z^λ_{pat} go back to [5, 17, 0].

We restrict ourselves to a finite or countable index set J. Therefore, we may assume that J is a subset of $\mathbb{N}$. We admit countably infinite sets J because of the beauty of the theory involved, and because of the fact that it may have applications to programming languages with dynamic process creation.

5.1. Patient vectors

Henceforward we will assume that set J is a nonempty subset of $\mathbb{N}$. Let a vector m be called *patient* if it satisfies the following two conditions:

(a) $\forall n \in \omega$: $\text{card}\{j \mid m_j < n\} \leq n$,

(b) $\forall r \in \omega\, \exists c \in \omega\, \forall n \in \omega$: $n > c \Rightarrow \text{card}\{j \mid m_j < n\} \leq n - r$.

Remarks. The vector m corresponds to the allowance counts of [5], and the priority variables of [0]. Both [5] and [0] contain condition (a). As far as we know, condition (b) is new. One notes that if set J is finite, then condition (b) holds for every vector m. In fact, if $n > r + \text{card}(J)$, then

$$\text{card}\{j \mid m_j < n\} \leq \text{card}(J) < n - r.$$

Example. The purpose of condition (b) can be explained as follows. Consider an infinite waiting queue of objects. Assume the objects have consecutive waiting numbers starting with zero. When the first object has been served, it re-enters the queue for a second service. As all waiting numbers have been given out, it will share its new waiting number with some other object of the queue. After a finite number of steps deadlock occurs since two objects cannot wait any longer.

More formally, let $A = \{a_j \mid j \in \mathbb{N}\}$ and $J = \mathbb{N}$ and $B_j = \{a_j\}$ where all actions a_j are different. Let Y be the transition system with one process y such that $s(y) = \{(a_j, y) \mid j \in \mathbb{N}\}$. In Z^ω we consider (y, m) where the vector m is given by $m_j = j$ for all j. Clearly, condition (a) is satisfied, and condition (b) is not. At (y, m) the only available action is a_0. Let the first transition be $a_0 : (y, m) \to (y, m')$ where m' satisfies $m'_j \leq j - 1$ for all $j > 0$, cf. Subsection 4.3. Assume that $m'_0 = r$. Vector m' does not satisfy condition (a) with $n = r + 1$. After at most r steps, a vector m'' is reached with $m''_j = 0$ and $m''_k = 0$ for certain indices $j \neq k$. By Subsection 4.3, the element (y, m'') in Z^ω has no transitions so that deadlock occurs.

This example shows that the sequence (m_j) must contain gaps in order to avoid deadlock. As the waiting queue must be re-entered infinitely many times, the queue must contain infinitely many gaps. This requirement is formalized in condition (b). The argument is generalized in the next result.

5.2.

Proposition 5.1. *Assume that (y, m) in Z^λ has an infinite execution path in Z^λ. Assume that the sets $B_j, j \in J$, are pairwise disjoint and that $Y(B_j) = Y$ for every index $j \in J$. Then vector m is patient.*

Proof. If $k \in J$, we put $C_k = A \setminus \bigcup_{j \neq k} B_j$. Since the sets B_j are pairwise disjoint, set A is the union of the sets C_k with $k \in J$. Let $\zeta = (a_i, (y^{i+1}, m^{i+1}))_{i \in \omega}$ be an infinite execution path of (y, m). We can choose a function $g : \omega \to J$ such that $a_i \in C_{g(i)}$ for every index i.

Let $a:(y, m)\to(y', m')$ be a transition in Z^λ with $a\in C_k$. If $j\in J$ differs from k, then $a\notin B_j$ and $y\in Y(B_j)$, and hence $m_j > m'_j$. This implies that

$$\forall t\in\omega:\quad \{j\mid m_j<t\}\subset\{k\}\cup\{j\mid m'_j<t-1\}.$$

By induction, it follows that the vectors (m^n) of the execution path ζ satisfy

$$\forall n, t\in\omega:\quad \{j\mid m_j<t\}\subset\{g(i)\mid i<n\}\cup\{j\mid m^n_j<t-n\}.$$

By specialization to the case $t=n$, we obtain

$$(*)\qquad \forall n\in\omega:\quad \{j\mid m_j<n\}\subset\{g(i)\mid i<n\}$$

This proves Condition 5.1(a).

As for Condition 5.1(b), let $r\in\omega$ be given. Choose a fixed index $h\in J$. Since the morphism $p_\lambda: Z^\lambda\to Y$ is T-fair, the image path $p_\lambda(\zeta)$ in Y is B_h-fair. Since $Y(B_h)=Y$, it follows that the set $\{i\mid a_i\in B_h\}$ is infinite. Therefore, we can choose $c\in\omega$ such that

$$\mathrm{card}\{i\mid i<c\wedge a_i\in B_h\}>r.$$

If $a_i\in B_h$, then $a_i\notin C_j$ for every index $j\neq h$, and hence, $g(i)=h$. It follows that

$$\mathrm{card}\{i\mid i<c\wedge g(i)=h\}>r.$$

Therefore, if $n>c$, it follows from formula (*) above that

$$\mathrm{card}\{j\mid m_j<n\}\leq\mathrm{card}\{g(i)\mid i<n\}\leq n-r.$$

This proves Condition 5.1(b). □

5.3.

Lemma 5.2. (a) *There exists a patient vector m in λ^J if and only if* $\mathrm{card}(J)\leq\lambda$.
(b) *If m is a patient vector in λ^J, there is a patient vector m' in ω^J such that $m_j\geq m'_j$ for all indices $j\in J$.*

Proof. (a): If λ is infinite, then $\mathrm{card}(J)\leq\omega\leq\lambda$. If λ is finite and m is a patient vector in λ^J, we have

$$\mathrm{card}(J)=\mathrm{card}\{j\mid m_j<\lambda\}\leq\lambda.$$

Conversely, assume that $\mathrm{card}(J)\leq\lambda$. Recall that J is a subset of $\mathbb{N}$. If J is finite, the vector m with $m_k=k$ for every index $k\in J$ is patient. If J is infinite, then λ is infinite and we use the vector m given by $m_k=2k$. This vector satisfies $\mathrm{card}\{j\mid m_j<n\}\leq\frac{1}{2}(n+1)$. Therefore, vector m is patient.

(b): Since vector m is patient we can construct an increasing function $h:\omega\to\omega$ such that $h(0)=0$ and

$$\forall r, n\in\omega:\quad n\geq h(r)\Rightarrow\mathrm{card}\{j\mid m_j<n\}\leq n-r.$$

Let $I=\{j\mid m_j\geq\omega\}$. Let vector m' in ω^J be defined by the conditions that $m'_j=m_j$ whenever $m_j<\omega$ and $m'_j=h(4j)$ for all indices $j\in I$. We have

$$\mathrm{card}\{j\mid m'_j<n\}=\mathrm{card}\{j\mid m_j<n\}+\mathrm{card}\{j\in I\mid h(4j)<n\}.$$

As function h is increasing, it follows that if $h(2r) \leq n < h(4r)$, then

$$\text{card}\{j \mid m'_j < n\} \leq (n-2r)+r = n-r.$$

This implies that if $n \geq h(2r)$, then $\text{card}\{j \mid m'_j < n\} \leq n-r$. Since $h(0)=0$, this proves that vector m' is patient. The inequalities are obvious. □

5.4.

Lemma 5.3. *Let m be a patient vector in λ^J. Let I be a nonempty subset of J. Then there is a patient vector m' in λ^J and an index $k \in I$ such that*

(a) *if $j \neq k$, then $m_j \geq m'_j$;*

(b) *if $j \in I \setminus \{k\}$, then $m_j > m'_j$.*

Proof. By Lemma 5.2(b), we may assume that $m_j < \omega$ for all indices $j \in J$. Recall that J is a nonempty subset of $\mathbb{N}$. We define

$$v = \min\{m_i \mid i \in I\}, \qquad k = \min\{i \in I \mid m_i = v\},$$

$$w = \max\{v\} \cup \{n \in \mathbb{N} \mid \text{card}\{j \mid m_j < n+1\} \geq n+1\},$$

$$m'_j = \text{if } j = k \to w$$

$$\quad [\!] \; j \neq k \wedge j \in I \to m_j - 1$$

$$\quad [\!] \; j \notin I \to m_j$$

$$\text{fi}.$$

Since m is patient, it follows from Condition 5.1(a) that $m'_j \geq 0$ for all indices j. By Condition 5.1(b), the number w is finite. If λ is finite, then $w < \lambda$ since $v < \lambda$ and $\text{card}(J) \leq \lambda$. This proves that $m' \in \lambda^J$. The conditions (a) and (b) hold. It remains to prove that m' is patient.

For the verification of Condition 5.1(a) we distinguish three cases.

Case 1: If $n < v$, then $\{j \mid m'_j < n\} = \{j \mid m_j < n\}$. By Condition 5.1(a) for vector m, it follows that

$$\text{card}\{j \mid m'_j < n\} \leq n.$$

Case 2: If $v \leq n \leq w$, then any index j with $m'_j < n$ satisfies $j \neq k$ and $m_j < n+1$. On the other hand, $m_k < n+1$. By Condition 5.1(a), this implies

$$\text{card}\{j \mid m'_j < n\} \leq \text{card}\{j \mid m_j < n+1\} - 1 \leq (n+1) - 1 = n.$$

Case 3: If $w < n$, then any $j \in J$ satisfies

$$(*) \qquad m'_j < n \;\Rightarrow\; m_j < n+1,$$

so that the definition of w implies

$$\text{card}\{j \mid m'_j < n\} \leq \text{card}\{j \mid m_j < n+1\} \leq n.$$

This proves that m' satisfies Condition 5.1(a). As for Condition 5.1(b), let $r \in \omega$ be given. By 5.1(b) for vector m, with $r := r+1$, there is a $c \in \omega$ such that

$$\forall n > c: \quad \mathrm{card}\{j \mid m_j < n\} \leq n-(r+1).$$

For any $n > \max\{w, c\}$, the implication (*) holds so that

$$\mathrm{card}\{j \mid m'_j < n\} \leq \mathrm{card}\{j \mid m_j < n+1\} \leq (n+1)-(r+1) = n-r.$$

Therefore, vector m' is patient. □

5.5. The patient T-fair implementation systems $Z^{\lambda}_{\mathrm{pat}}$

We define $Z^{\lambda}_{\mathrm{pat}}$ to be the full subsystem of Z^{λ} that consists of the pairs (y, m) in Z^{λ} such that vector m is patient.

Theorem 5.4. *If $\lambda \geq \mathrm{card}(J)$, then the restriction $p_\lambda : Z^{\lambda}_{\mathrm{pat}} \to Y$ is T-fair, surjective, and deadlock-free.*

(b) *If $f : X \to Y$ is a T-fair morphism, there is an ordinal number λ and a morphism $h : X \to Z^{\lambda}_{\mathrm{pat}}$ such that $f = p_\lambda \circ h$.*

Proof. (a): Since $p_\lambda : Z^{\lambda} \to Y$ is a T-fair morphism, it follows with Definition 4.1 that the restriction to $Z^{\lambda}_{\mathrm{pat}}$ is T-fair. The surjectivity of the restriction follows from Lemma 5.2(a).

It remains to be shown that the restriction is deadlock-free. Let $z \in Z^{\lambda}_{\mathrm{pat}}$ be such that $p_\lambda(z)$ is nonempty. Write $z = (y, m)$. Then y is nonempty in Y and that m is a patient vector. Put $I = \{j \in J \mid y \in Y(B_j)\}$. If I is nonempty, we choose m' and k as specified in Lemma 5.3. Since $y \in Y(B_k)$, we can choose a transition $a : y \to y'$ in Y with $a \in B_k$. Then we have a transition $a : (y, m) \to (y', m')$ in $Z^{\lambda}_{\mathrm{pat}}$ so that the process $z = (y, m)$ is nonempty. If set I is empty, we use that the process y is nonempty and we choose an arbitrary transition $a : y \to y'$. Then we have the transition $a : (y, m) \to (y', m)$ in $Z^{\lambda}_{\mathrm{pat}}$ so that process $z = (y, m)$ is nonempty. This proves that the restriction $p_\lambda : Z^{\lambda}_{\mathrm{pat}} \to Y$ is deadlock free.

(b): By Theorem 4.1, there is an ordinal number λ and a morphism $h' : X \to Z^{\lambda}$ with $f = p_\lambda \circ h'$. Since $Z^{\lambda} \subset Z^{\mu}$ whenever $\lambda < \mu$, we may assume that $\lambda \geq \omega$. By Lemma 5.2(a), we can choose a patient vector $q \in \omega^J$. If m is an arbitrary vector in λ^J, we define the sum vector $q+m$ by $(q+m)_j = q_j + m_j$. If $n < \omega$ and $\mu \geq \omega$, then $n + \mu = \mu$, cf. [19, Section 8.10]. It follows that $q+m \in \lambda^J$. As vector q is patient, the sum vector $q+m$ is also patient. Let the translation $q_+ : Z^{\lambda} \to Z^{\lambda}_{\mathrm{pat}}$ be the function defined by $q_+(y, m) = (y, q+m)$. It is easy to see that q_+ is a morphism of transition systems with $p_\lambda \circ q_+ = p_\lambda$. It follows that $h = q_+ \circ h'$ is a morphism $X \to Z^{\lambda}_{\mathrm{pat}}$ with $f = p_\lambda \circ h$. □

Example. For Example 5.1 we may obtain the following infinite fair execution path in Z^ω. Let the successor of a vector $m = (m_k)_{k \in \mathbb{N}}$ be defined by

$$\begin{aligned} \mathrm{succ}(m)_k = \ & \textbf{if } m_k = 0 \to \min\{x \in \mathbb{N} \mid \forall r: x \neq m_r - 1\} \\ & \| \ m_k > 0 \to m_k - 1 \\ & \textbf{fi}. \end{aligned}$$

Let the sequence of vectors $(m^i)_{i \in \mathbb{N}}$ be given by $m^{i+1} = \mathrm{succ}(m^i)$ where m^0 is given by $m^0_j = 2j$, as in the proof of Lemma 5.2(a). By induction one proves that for all $i \in \mathbb{N}$ there is precisely one index $k = r(i)$ such that $m_k = 0$. It follows that Z^ω contains an infinite fair execution path with transitions $a_{r(i)}: (y, m^i) \to (y, m^{i+1})$.

5.6. Remark

Around 1971, Dijkstra determined a class of allocation strategies for a finite set of processes such that each process has only bounded delay, cf. [5]. He used the scenario of Construction 4.3 above, with fixed bounds on the components m_j. He used the word "safety" for Condition 5.1(a).

Following [17, 0], etc., the present paper admits unbounded but finite delays. In [17, p. 420], Plotkin proposed a generative operational semantics for a weakly fair merge of two processes. The patience of the delayed process was encoded in the operator symbols. His Theorem 1 expressed that all fair execution sequences were obtained. In [0, Lemma 3], this result is generalized to finitely many processes with a scenario analogous to Construction 4.3. The paper [0] also contains the result for strong fairness. Theorem 5.4 can be viewed as an abstract version of this result, in the case of strong fairness with possibly infinitely many processes. Strictly speaking, however, these results are more akin to Theorem 5.6 below.

5.7.

Corollary 5.5. *Assume* $\mathrm{card}(J) \leq \lambda$.

(a) *The subsystem* Z^λ_{pat} *is contained in the living subsystem* Z^λ_{liv} *of* Z^λ *with respect to the morphism* p_λ, *cf. Definition* 1.6. *The restriction* $p_\lambda: Z^\lambda_{\mathrm{liv}} \to Y$ *is surjective.*

(b) *Assume that the sets* B_j, $j \in J$, *are pairwise disjoint and that* $Y(B_j) = Y$ *for every index* $j \in J$. *Then* $Z^\lambda_{\mathrm{pat}} = Z^\lambda_{\mathrm{liv}}$.

Proof. (a): This follows from Theorem 5.4 and Definition 1.6.

(b): In view of part (a), it suffices to prove that Z^λ_{liv} is contained in Z^λ_{pat}. Let (y, m) be element of Z^λ_{liv}. Choose an execution path ζ of (y, m) in Z^λ such that the image path $p_\lambda(\zeta)$ is a maximal path in Y, cf. Definition 1.6. Since there is a $j \in J$ with $Y(B_j) = Y$, there is no empty process in Y. Therefore, path $p_\lambda(\zeta)$ is infinite. It follows that path ζ is infinite so that vector m is patient by Proposition 5.1. This proves that $(y, m) \in Z^\lambda_{\mathrm{pat}}$. □

5.8.

Theorem 5.6. *Let $(a_i, y_{i+1})_{i\in\omega}$ be a T-fair execution path of a process y_0 in the system Y. In the transition system $Z^{\omega}_{\mathrm{pat}}$ there is a process z_0 with an execution path $(a_i, z_{i+1})_{i\in\omega}$ such that $p_\omega(z_i) = y_i$ for every index $i \in \omega$.*

Proof. We first use Fact 3.3 to obtain, in the system Z^{ω}, a process z'_0 and an execution path $(a_i, z'_{i+1})_{i\in\omega}$ such that $p_\omega(z'_i) = y_i$ for every index $i \in \omega$. The method is the same as in the proof of Theorem 4.1(b). Then we argue as in the proof of Theorem 5.4(b). We choose a patient vector $q \in \omega^J$, and we use the translation $q_+ : Z^{\omega} \to Z^{\omega}_{\mathrm{pat}}$ as defined in the proof of Theorem 5.4(b) to construct $z_i = q_+(z'_i)$. The verifications are easy. □

6. The fair communicating merge

6.0.

In this section we show that a fair communicating merge of a finite or countably infinite family of processes can be treated as a direct application of the theory we have developed.

6.1. Preparation of the action symbols

Let $(x_j)_{j\in J}$ be a family of processes. Each component process x_j is element of some transition system X_j. So we have a family of transition systems $(X_j)_{j\in J}$. Let the actions of the various components be labeled with a component identification. Let B_j denote the set of symbols of actions of the component X_j. So we have

$$\forall x \in X_j \, \forall (a, y) \in s(x): \quad a \in B_j \wedge y \in X_j.$$

We admit synchronous communications between different components. A communication between two components is labeled with identifications of sender and receiver. Therefore, we can use the same action symbol in both components. Every symbol in an intersection $B_j \cap B_k$ with $j \neq k$ stands for such a communication. If i, j, k are three different indices, the intersection $B_i \cap B_j \cap B_k$ is empty (the hand-shaking axiom of [3]). Finally, by changing A we may assume that A is the union of the sets B_j with $j \in J$. Now every symbol $a \in A$ belongs to at least one and at most two subsets B_j.

6.2. Demonic communicating merge

The family of processes $x = (x_j)_{j\in J}$ can be considered as an element of the cartesian product set $W = \prod_j X_j$. If the set W is made into a transition system, the family x becomes a process. We give the set W a transition system structure such that process x is the demonic communicating merge of the processes x_j. The transition function s of W is defined as follows: if $x \in W$, then $s(x)$ is the set of the pairs (a, y) with $a \in A$ and $y \in W$ such that

$$\forall j \in J: \quad (a \in B_j \wedge (a, y_i) \in s(x_j)) \vee (a \notin B_j \wedge y_i = x_j).$$

To justify that a process $x \in W$ is the demonic communicating merge of its component processes x_j, we note that for any action $a: x \to y$ in W there are one or two indices j with $a \in B_j$. Therefore, one or two component processes x_j are involved in the transition. The other components x_j are unaffected. Two components are involved if and only if a hand-shaking communication occurs.

Remark. This way to model the communicating merge was inspired by the weaving operator of trace structures, as introduced by Van de Snepscheut in [18].

6.3. Fair implementations of the communicating merge

In the transition system W of the merged families $(x_j)_{j \in J}$, the component j is involved in a transition $a: x \to y$ if and only if $a \in B_j$. Therefore, fairness of execution paths in system W with respect to the family $T = (B_j)_{j \in J}$ is the same thing as (strong) fairness with respect to the operands of the merge.

Under the assumption that the set J is nonempty, and finite or countably infinite, the theory of Section 5 provides a family of T-fair, surjective, and deadlock-free morphisms $p_\lambda : Z^\lambda_{\text{pat}} \to W$, indexed by ordinal numbers $\lambda \geq \text{card}(J)$. The T-fairness of p_λ means that the implementation provides a scheduling such that the components x_j of process x will not suffer individual starvation if that can be avoided infinitely many times. Inside of a component, however, there is no fairness guarantee.

The fact that $p_\lambda : Z^\lambda_{\text{pat}} \to W$ is deadlock-free says that the fair implementation does not introduce deadlock. It can happen of course that every component x_j is waiting for some communication that is not available. Such a kind of deadlock also occurs in the transition system W of the demonic merges. It is not deadlock of morphisms in our sense. For there are no morphisms of transition systems from W to the component systems X_j.

The universality property of Theorem 5.4(b) can be understood as follows: if X is an arbitrary fair implementation of the merge W, then there is an ordinal number λ such that X is an implementation of Z^λ_{pat}. All observations of X can be interpreted as observations of Z^λ_{pat}. In particular, it cannot be falsified that the implementation X makes hidden predictive choices of the maximal number of enabled states that a given component can wait.

Acknowledgment

The main inspiration came by reading [2]. The work was stimulated further by discussions in Amsterdam with De Bakker, Bergstra, and others. Coen Bron provided the reference [5]. I am indebted to the referees for good suggestions and valuable criticisms, and it is due to their insistence that the paper contains some diagrams.

References

[0] K.R. Apt and E.-R. Olderog, Proof rules and transformations dealing with fairness, *Sci. Comput. Programming* 3 (1983) 65–100.

[1] K.R. Apt, A. Pnueli and J. Stavi, Fair termination revisited—with delay, *Theoret. Comput. Sci.* **33** (1984) 65–84.

[2] J.W. de Bakker and J.I. Zucker, Processes and a fair semantics for the ADA rendez-vous, in: *Proc. 10th ICALP*, Lecture Notes in Computer Science **154** (Springer, Berlin, 1983) 52–66.

[3] J.A. Bergstra and J.W. Klop, Algebra of communicating processes, in: J.W. de Bakker, M. Hazewinkel and J.K. Lenstra, eds., *Proc. CWI Symp. on Mathematics and Computer Science* (North-Holland, Amsterdam, 1986).

[4] G. Costa and C. Stirling, Weak and strong fairness in CCS, in: *Proc. Conf. on Mathematical Foundations of Computer Science 1984*, Lecture Notes in Computer Science **176** (Springer, Berlin, 1984) 245–254.

[5] E.W. Dijkstra, A class of allocation strategies inducing bounded delays only, Tech. Rept., Tech. Univ. Eindhoven, EWD 319, 1971.

[6] N. Francez, *Fairness* (Springer, Berlin, 1986).

[7] O. Grümberg, N. Francez, J.A. Makowsky and W.P. de Roever, A proof rule for fair termination of guarded commands, in: J.W. de Bakker and J.C. van Vliet, eds., *Algorithmic Languages* (North-Holland, Amsterdam, 1981) 399–416.

[8] W.H. Hesselink, Nondeterminism in data types, a mathematical approach, *ACM Trans. on Programming Languages and Systems* **10** (1988).

[9] W.H. Hesselink, Interpretations of recursion under unbounded nondeterminacy, *Theoret. Comput. Sci.* **59** (1988) 211–234 (this issue).

[10] W.H. Hesselink, The universal transition system for weak bisimulation, in preparation.

[11] D. Kapur, Towards a theory of abstract data types, Ph.D. Thesis, Mass. Institute of Technology, Tech. Rept. MIT/LCS/TR-237.

[12] R. Keller, Formal verification of parallel programs, *J. ACM* **19** (1976) 371–384.

[13] D. Lehmann, A. Pnueli and J. Stavi, Impartiality, justice and fairness: the ethics of concurrent termination, in: *Proc. 8th ICALP*, Lecture Notes in Computer Science **115** (Springer, Berlin, 1981) 264–277.

[14] R. Milner, Lectures on a calculus for communicating systems, in: *Proc. Seminar on Concurrency*, Lecture Notes in Computer Science **197** (Springer, Berlin, 1985) 197–220.

[15] S. Owicki and L. Lamport, Proving liveness properties of concurrent programs, *ACM Trans. on Programming Languages and Systems* **4** (1982) 455–495.

[16] D. Park, On the semantics of fair parallelism, in: D. Bjørner, ed., *Abstract Software Specifications*, Lecture Notes in Computer Science **86** (Springer, Berlin, 1980) 504–526.

[17] G.D. Plotkin, A powerdomain for countable nondeterminism, in: *Proc. 9th ICALP*, Lecture Notes in Computer Science **140** (Springer, Berlin, 1982) 418–428.

[18] J.L.A. van de Snepscheut, *Trace Theory and VLSI Design*, Lecture Notes in Computer Science **200** (Springer, Berlin, 1985).

[19] G. Takeuti and W.M. Zaring, *Introduction to Axiomatic Set Theory*, Graduate Texts in Mathematics **1** (Springer, Berlin, 1971).